Applications, Solvers, and Utilities

- OpenFOAM is first and foremost a *C++ library*, used primarily to create executables, known as *applications*. The applications fall into two categories: *solvers*, that are each designed to solve a specific continuum mechanics problem; and *utilities*, that are designed to perform tasks that involve data manipulation.
- OpenFOAM is distributed with a large number of applications, but soon any advanced user will start developing new applications for his/ her special needs. The basic way to do this is to find and copy an application that almost does what is needed, and then to modify it by copy/paste from other applications that has some features that are needed.
- Special applications for pre- and post-processing are included in OpenFOAM. Converters to/from other pre- and post-processors are available.

Before we go on ...

- Some useful aliases:

```
app       cd $FOAM_APP
foam      cd $WM_PROJECT_DIR
foamfv    cd $FOAM_SRC/finiteVolume
foamsrc   cd $FOAM_SRC/$WM_PROJECT
lib       cd $FOAM_LIB
run       cd $FOAM_RUN
sol       cd $FOAM_SOLVERS
src       cd $FOAM_SRC
tut       cd $FOAM_TUTORIALS
util      cd $FOAM_UTILITIES
```

  Type `alias` for a complete list

- We will have a look at environment variables when looking at installation of OpenFOAM. Find out what an environment variable means by: `echo $FOAM_APP`. See all the environment variables by: `env`.

Finding the source code of the applications in OpenFOAM

- The source code for the applications is arranged in a structure that is useful for finding the application you need.
- Use the pre-defined alias `app` to go to the applications directory: `$WM_PROJECT_DIR/applications`
- You will find: `Allwmake bin solvers test utilities`
- `Allwmake` is used to compile all the applications.
- `bin` contains the binaries of the applications after compilation.
- `solvers` contains the source code of the solvers.
- `utilities` contains the source code of the utilities.
- `test` contains source code for testing specific features of OpenFOAM.

Solvers in OpenFOAM-1.6.x

- In `$WM_PROJECT_DIR/applications/solvers` you find the source code for the solvers arranged according to:

```
basic          discreteMethods    financial        lagrangian
combustion     DNS                heatTransfer     multiphase
compressible   electromagnetics   incompressible   stressAnalysis
```

- In `$WM_PROJECT_DIR/applications/solvers/incompressible` you find the solver source code directories:

```
boundaryFoam   nonNewtonianIcoFoam   pisoFoam
channelFoam    pimpleDyMFoam         shallowWaterFoam
icoFoam        pimpleFoam            simpleFoam
```

- Inside each solver directory you find a `*.C` file with the same name as the directory. This is the main file, where you will find the top-level source code and a short description of the solver. For `icoFoam`:

```
Transient solver for incompressible, laminar flow of
Newtonian fluids.
```

For a more complete description, you have the source code right there.

Utilities in OpenFOAM-1.6.x

- In `$WM_PROJECT_DIR/applications/utilities` you find the source code for the utilities arranged according to:

```
errorEstimation   parallelProcessing   surface
mesh              postProcessing       thermophysical
miscellaneous     preProcessing
```

- In `$WM_PROJECT_DIR/applications/utilities/postProcessing/velocityField` you find:

```
Co          flowType   Mach   Q               uprime
enstrophy   Lambda2    Pe     streamFunction  vorticity
```

- Inside each utility directory you find a `*.C` file with the same name as the directory. This is the main file, where you will find the top-level source code and a short description of the utility. For `vorticity`:

```
Calculates and writes the vorticity of velocity field U.
The -noWrite option just outputs the max/min values without writing
    the field.
```

Additional solvers and utilities

See the Wiki:

`http://www.openfoamwiki.net`

the Forum:

`http://www.cfd-online.com/Forums/openfoam/`

and the OpenFOAM-extend project at SourceForge:

`http://sourceforge.net/projects/openfoam-extend/`

Finding tutorials for the applications in OpenFOAM

- Use the pre-defined alias `tut` to go to the tutorials directory: `$WM_PROJECT_DIR/tutorials`, where there are complete set-ups of cases for all the solvers.

- Note that it is recommended to copy the tutorials to your `$WM_PROJECT_USER_DIR/run` directory before running them or making any modification to them, so that you always have a clean version of the tutorials.

- There are no specific tutorials for the utilities, but some of the solver tutorials also show how to use the utilities. We will have a look at some of them while we run through the `icoFoam cavity` tutorial.

- There are only written tutorials to some of the applications and utilities in OpenFOAM.

- The purpose of your project is to add written tutorials to some part of OpenFOAM. See the tutorials in the UserGuide, ProgrammersGuide, and projects from previous years as examples.

The icoFoam/cavity tutorial

- Basic procedure when running a tutorial, in this case `icoFoam/cavity`:

  ```
  cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity $FOAM_RUN
  run
  cd cavity
  ```

  You have copied the `cavity` tutorial and moved to `$FOAM_RUN/cavity`

- The mesh is defined by a dictionary that is read by the `blockMesh` utility

  ```
  blockMesh
  ```

  You have now generated the grid in OpenFOAM format. Check the output from blockMesh!

- Check the mesh by

  ```
  checkMesh
  ```

  You see the grid size, the geometrical size and some grid checks (e.g. cell volumes).

- This is a case for the `icoFoam` solver, so run

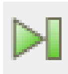
  ```
  icoFoam >& log&
  ```

  You now ran the simulation in background using the settings in the case, and forwarded the errors and standard output to the `$FOAM_RUN/cavity/log` file, where the Courant numbers and the residuals are shown.

Application parameters

- Most OpenFOAM applications take parameters. Use the `-help` flag to get info:
  `blockMesh -help`
  yields:
  ```
  Usage: blockMesh [-dict dictionary] [-case dir]
  [-blockTopology] [-region name]  [-help] [-doc] [-srcDoc]
  ```
- The `[-case dir]` is the most common one, and with that you can specify the path to the case directory if you do not want to move to that case directory.

Post-process the icoFoam/cavity tutorial

- View the results using paraFoam:
  `paraFoam`
  Click on 'Last Frame': 
  Click Accept.
  Color by Pressure using `Display/Color by` 

  Move, rotate and scale the visualization using the mouse

- We will learn how to use paraFoam more further on.

- Exit paraFoam: `File/Exit`

- The results may also be viewed using third-party products:
  `foamToEnsight` etc., type: `foamTo[TAB]` to see alternatives.
  There is also a direct reader for Ensight - see the UserGuide.

- For post-processing in Fluent, run:
  `foamMeshToFluent`, and `foamDataToFluent` (`controlDict` is used to specify the time step, and a `foamDataToFluentDict` dictionary is required - see the UserGuide).

icoFoam/cavity tutorial - What did we do?

- We will have a look at what we did when running the `cavity` tutorial by looking at the case files.
- First of all it should be noted that `icoFoam` is a *Transient solver for incompressible, laminar flow of Newtonian fluids*
- The case directory originally contains the following sub-directories: `0`, `constant`, and `system`. After our run it also contains the output `0.1`, `0.2`, `0.3`, `0.4`, `0.5`, and log
- The `0*` directories contain the values of all the variables at those time steps. The `0` directory is thus the initial condition.
- The `constant` directory contains the mesh and a `transportProperties` dictionary for the kinematic viscosity.
- The `system` directory contains settings for the run, discretization schemes, and solution procedures.
- The `icoFoam` solver reads the files in the case directory and runs the case according to those settings.

icoFoam/cavity tutorial - The constant directory

- The `constant/transportProperties` file is a dictionary for the dimensioned scalar `nu`.
- The `polyMesh` directory originally contains the `blockMeshDict` dictionary for the `blockMesh` grid generator, and now also the mesh in OpenFOAM format.
- We will now have a quick look at the `blockMeshDict` dictionary in order to understand what grid we have used.

icoFoam/cavity tutorial - blockMeshDict dictionary

- The blockMeshDict dictionary first of all contains a number of vertices:

```
convertToMeters 0.1;
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

- There are eight vertices defining a 3D block. OpenFOAM always uses 3D grids, even if the simulation is 2D.
- `convertToMeters 0.1;` multiplies the coordinates by 0.1.

icoFoam/cavity tutorial - blockMeshDict dictionary

- The blockMeshDict dictionary secondly defines a block and the mesh from the vertices:

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);
```

- `hex` means that it is a structured hexahedral block.
- `(0 1 2 3 4 5 6 7)` is the vertices used to define the block. The order of these is important - they should form a right-hand system! See the UserGuide.
- `(20 20 1)` is the number of grid *cells* in each direction.
- `simpleGrading (1 1 1)` is the expansion ratio, in this case equidistant. The numbers are the ratios between the end cells along three edges. See the UserGuide.

icoFoam/cavity tutorial - blockMeshDict dictionary

- The blockMeshDict dictionary finally defines three patches:

```
patches
(
    wall movingWall
    (
        (3 7 6 2)
    )
    wall fixedWalls
    (
        (0 4 7 3)
        (2 6 5 1)
        (1 5 4 0)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);
```

icoFoam/cavity tutorial - blockMeshDict dictionary

- Each patch defines a type, a name, and a list of boundary faces
- Let's have a look at the fixedWalls patch:

```
wall fixedWalls
(
    (0 4 7 3)
    (2 6 5 1)
    (1 5 4 0)
)
```

- `wall` is the type of the boundary.
- `fixedWalls` is the name of the patch.
- The patch is defined by three sides of the block according to the list, which refers to the vertex numbers. The order of the vertex numbers is such that they are marhced clock-wise when looking from inside the block. Thi is important, and unfortunately `checkMesh` will not find such problems.

icoFoam/cavity tutorial - blockMeshDict dictionary

- There are two empty sub-dictionaries in the `icoFoam/cavity` tutorial:

```
edges();
mergePatchPairs();
```

- `edges();` is used to define shapes of the edges if they are not straight - `polySpline`, `polyLine`, `line`, `simpleSpline`, `arc`. We will use `arc` later on.
- `mergePatchPairs();` is used to stitch two blocks that are not connected, but share the same physical surface at a patch of each block. This means that both blocks have a patch which is defined with four vertices in the same location as the corresponding patch in the neighbouring block, but the vertices are not the same in both blocks. It should be possible to stitch non-conformal meshes so the number of nodes and the distribution of the nodes do not have to be the same on both sides.

icoFoam/cavity tutorial - blockMeshDict dictionary

- To sum up, the blockMeshDict dictionary generates a block with:
  x/y/z dimensions 0.1/0.1/0.01
  $20 \times 20 \times 1$ cells
  `wall fixedWalls` patch at three sides
  `wall movingWall` patch at one side
  `empty frontAndBack` patch at two sides

- The type `empty` tells OpenFOAM that it is a 2D case, i.e. the equations will not be solved for in the direction of the `empty` patches.

- Read more about `blockMesh` yourself in the UserGuide.

- You can also convert mesh files from third-party products - see the User-Guide. If you use ICEM, a good procedure is to write a Fluent mesh file (`*.msh`) and convert it with `fluentMeshToFoam` or `fluent3DMeshToFoam`.

icoFoam/cavity tutorial - the mesh

- `blockMesh` uses the `blockMeshDict` to generate some files in the `constant/polyMesh` directory:

```
boundary   faces   neighbour   owner   points
```

- `boundary` shows the definitions of the patches, for instance:

```
movingWall
{
    type wall;
    nFaces 20;
    startFace 760;
}
```

- The other files defines the points, faces, and the relations between the cells.

icoFoam/cavity tutorial - The system directory

- The `system` directory consists of three set-up files:

  `controlDict   fvSchemes   fvSolution`

- `controlDict` contains general instructions on how to run the case.
- `fvSchemes` contains instructions on which discretization schemes that should be used for different terms in the equations.
- `fvSolution` contains instructions on how to solve each discretized linear equation system. It also contains instructions for the PISO pressure-velocity coupling.

icoFoam/cavity tutorial - The controlDict dictionary

- The `controlDict` dictionary consists of the following lines:

```
application         icoFoam;
startFrom           startTime;
startTime           0;
stopAt              endTime;
endTime             0.5;
deltaT              0.005;
writeControl        timeStep;
writeInterval       20;
purgeWrite          0;
writeFormat         ascii;
writePrecision      6;
writeCompression    uncompressed;
timeFormat          general;
timePrecision       6;
runTimeModifiable yes;
```

icoFoam/cavity tutorial - The controlDict dictionary

- `application icoFoam;`
  Was previously used to tell the GUI `FoamX` in OpenFOAM-1.4.1 and earlier to use the set-up specifications of the `icoFoam` solver. I don't know if it is used for any other purpose. Please help me find out!

- The following lines tells `icoFoam` to start at `startTime`=0, and stop at `endTime`=0.5, with a time step `deltaT`=0.005:

```
startFrom         startTime;
startTime         0;
stopAt            endTime;
endTime           0.5;
deltaT            0.005;
```

icoFoam/cavity tutorial - The controlDict dictionary

- The following lines tells `icoFoam` to write out results in separate directories (`purgeWrite 0;`) every 20 `timeStep`, and that they should be written in `uncompressed ascii` format with `writePrecision` 6. `timeFormat` and `timePrecision` are instructions for the names of the time directories.

```
writeControl        timeStep;
writeInterval       20;
purgeWrite          0;
writeFormat         ascii;
writePrecision      6;
writeCompression    uncompressed;
timeFormat          general;
timePrecision       6;
```

  I recommend the use of `compressed ascii` format, which does not fill up your hard drive, and you can still open the files with `vim`.

- `runTimeModifiable yes;` allows you to make modifications to the case while it is running.

Specifying a maximum Courant number and varying time steps

- Some solvers, like the `interFoam` solver allows a varying time step, based on a maximum Courant number. Some extra entries should then be added to the controlDict dictionary:

```
adjustTimeStep  yes;
maxCo           0.5;
maxDeltaT       1;
```

- The solver is told to adjust the time step so that the output still occurs at specific times using:

```
writeControl    adjustableRunTime;
writeInterval   0.05;
```

icoFoam/cavity tutorial - A dictionary hint

- If you don't know which entries are available for a specific key word in a dictionary, just use a dummy and the solver will list the alternatives, for instance:

  ```
  stopAt            dummy;
  ```

  When running icoFoam you will get the message:

  ```
  dummy is not in enumeration
  4
  (
  endTime
  writeNow
  noWriteNow
  nextWrite
  )
  ```

  and you will know the alternatives.

icoFoam/cavity tutorial - More dictionary hints

- You may also use C++ commenting in the dictionaries:

```
// This is my comment
/* My comments, line 1
   My comments, line 2 */
```

- Dictionary expansion mechanism:
  - Include another file:
    ```
    #include "boundaryConditions"
    ```
  - Define parameters:
    ```
    velocity1       1;
    ```
  - Use parameters:
    ```
    $velocity1
    ```

icoFoam/cavity tutorial - The fvSchemes dictionary

- The `fvSchemes` dictionary defines the discretization schemes, in particular the time marching scheme and the convections schemes:

```
ddtSchemes
{
    default          Euler;
}
divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
}
```

- Here we use the `Euler` implicit temporal discretization, and the `linear` (central-difference) scheme for convection.
- `default none;` means that schemes must be explicitly specified.
- Find the available convection schemes using a 'dummy' dictionary entry. There are 58 alternatives, and the number of alternatives are increasing!

icoFoam/cavity tutorial - The fvSolution dictionary

- The `fvSolution` dictionary defines the solution procedure.
- The solutions of the p linear equation systems is defined by:

```
p
{
        solver           PCG;
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
}
```

- The p linear equation system in solved using the Conjugate Gradient solver `PCG`, with the preconditioner `DIC`.
- The solution is considered converged when the residual has reached the `tolerance`, or if it has been reduced by `relTol` at each time step.
- `relTol` is here set to zero since we use the PISO algorithm. The PISO algorithm only solves each equation once per time step, and we should thus solve the equations to `tolerance 1e-06` at each time step. `relTol 0;` disables `relTol`.

icoFoam/cavity tutorial - The fvSolution dictionary

- The solutions of the U linear equation systems is defined by:

```
U
{
    solver           PBiCG;
    preconditioner   DILU;
    tolerance        1e-05;
    relTol           0;
}
```

- The U linear equation system in solved using the Conjugate Gradient solver `PBiCG`, with the preconditioner `DILU`.
- The solution is considered converged when the residual has reached the `tolerance 1e-05` for each time step.
- `relTol` is again set to zero since we use the PISO algorithm. `relTol 0;` disables `relTol`.

icoFoam/cavity tutorial - The fvSolution dictionary

- The settings for the PISO algorithm are specified in the `PISO` entry:

```
PISO
{
    nCorrectors      2;
    nNonOrthogonalCorrectors 0;
    pRefCell         0;
    pRefValue        0;
}
```

- `nCorrectors` is the number of PISO correctors. You can see this in the log file since the p equation is solved twice, and the pressure-velocity coupling is thus done twice.
- `nNonOrthogonalCorrectors` adds corrections for non-orthogonal grids, which may sometimes influence the solution.
- The pressure is set to `pRefValue 0` in cell number `pRefCell 0`. This is over-ridden if a constant pressure boundary condition is used for the pressure.

icoFoam/cavity tutorial - The 0 directory

- The `0` directory contains the dimensions, and the initial and boundary conditions for all primary variables, in this case `p` and `U`. `U`-example:

```
dimensions        [0 1 -1 0 0 0 0];
internalField     uniform (0 0 0);
boundaryField
{   movingWall
    {
        type                fixedValue;
        value               uniform (1 0 0);
    }
    fixedWalls
    {
        type                fixedValue;
        value               uniform (0 0 0);
    }
    frontAndBack
    {
        type                empty;
    }}
```

icoFoam/cavity tutorial - The 0 directory

- `dimensions [0 1 -1 0 0 0 0];` states that the dimension of `U` is m/s. We will have a further look at this later on.
- `internalField uniform (0 0 0);` sets `U` to zero internally.
- The boundary patches `movingWall` and `fixedWalls` are given the `type fixedValue; value uniform (1 0 0);` and `(0 0 0)` respectively, i.e. $U_x = 1m/s$, and $U = 0m/s$ respectively.
- The `frontAndBack` patch is given `type empty;`, indicating that no solution is required in that direction since the case is 2D.
- You should now be able to understand `0/p` also.
- The resulting 0.* directories are similar but the `internalField` is now a `nonuniform List<vector>` containing the results. Some boundary condition types also give `nonuniform List`. There is also a `phi` file, containing the resulting face fluxes that are needed to give a perfect restart. There is also some time information in `0.*/uniform/time`. The `0.*/uniform` directory can be used for uniform information in a parallel simulation.

icoFoam/cavity tutorial - The log file

- If you followed the earlier instructions you should now have a `log` file. That file contains mainly the Courant numbers and residuals at all time steps:

```
Time = 0.09

Courant Number mean: 0.116099 max: 0.851428
DILUPBiCG:  Solving for Ux, Initial residual = 0.000443324,
                    Final residual = 8.45728e-06, No Iterations 2
DILUPBiCG:  Solving for Uy, Initial residual = 0.000964881,
                    Final residual = 4.30053e-06, No Iterations 3
DICPCG:  Solving for p, Initial residual = 0.000987921,
                    Final residual = 5.57037e-07, No Iterations 26
time step continuity errors : sum local = 4.60522e-09,
                    global = -4.21779e-19, cumulative = 2.97797e-18
DICPCG:  Solving for p, Initial residual = 0.000757589,
                    Final residual = 3.40873e-07, No Iterations 26
time step continuity errors : sum local = 2.81602e-09,
                    global = -2.29294e-19, cumulative = 2.74868e-18
ExecutionTime = 0.08 s  ClockTime = 0 s
```

icoFoam/cavity tutorial - The log file

- Looking at the `Ux` residuals

```
DILUPBiCG:  Solving for Ux, Initial residual = 0.000443324,
                    Final residual = 8.45728e-06, No Iterations 2
```

- We see that we used the `PBiCG` solver with `DILU` preconditioning.
- The `Initial residual` is calculated before the linear equation system is solved, and the `Final residual` is calculated afterwards.
- We see that the `Final residual` is less than our `tolerance` in `fvSolution` (`tolerance 1e-05;`).
- The `PBiCG` solver used 2 iterations to reach convergence.
- We could also see in the log file that the pressure residuals and continuity errors were reported twice each time step. That is because we specified `nCorrectors 2;` for the `PISO` entry in `fvSolution`.
- The `ExecutionTime` is the elapsed CPU time, and the `ClockTime` is the elapsed wall clock time for the latest time step (approximate!!!).

icoFoam/cavity tutorial - The log file

- It is of interest to have a graphical representation of the residual development.
- The `foamLog` utility is basically a script using `grep`, `awk` and `sed` to extract values from a log file. See `$WM_PROJECT_DIR/bin/foamLog` for the source code.
- `foamLog` uses a database (`foamLog.db`) to know what to extract. The `foamLog.db` database can be modified if you want to extract any other values that `foamLog` doesn't extract by default. (`find $WM_PROJECT_DIR -iname "*foamLog.db*"` and make your own copy to modify in `$HOME/.foamLog.db`, which will be used automatically)
- `foamLog` is executed on the `cavity` case with log-file `log` by:
  `foamLog log`
- A directory `logs` has now been generated, with extracted values in ascii format in two columns. The first column is the `Time`, and the second column is the value at that time.
- Type `foamLog -h` for more information.
- The graphical representation is then given by Matlab, `xmgrace -log y Ux_0 p_0` or `gnuplot`: `set logscale y`, `plot "Ux_0","Uy_0","p_0"`.
- We will later also use user-contributed `pyFoam` to plot residuals on-the-fly.

icoFoam/cavity tutorial - The icoFoam solver

- The `icoFoam` solver source code is located in
  `$WM_PROJECT_DIR/applications/solvers/incompressible/icoFoam`
  where you can find two files, `createFields.H` and `icoFoam.C`, and a `Make` directory. (There is also a `icoFoam.dep` file, which is generated when compiling)

- The `Make` directory contains two files, `files` and `options`, that specifies how `icoFoam` should be compiled. We will have a look at that later. (The `linux*` directories are generated when compiling)

- In `icoFoam.C` you basically see a `runTime` loop, the specification and solution of the `UEqn` coefficient matrix, and the `PISO` loop.

- In `createFields.H` the kinematic viscosity, the velocity field, and the pressure fields are read from the `startTime` directory. The face convections, `phi` are computed if they are not available in the `startTime` directory. Finally, the reference pressure is set if there are no constant pressure boundary conditions.

- We will study these files further later on, and we will learn how to copy a solver, modify it slightly, compile, and run it.